RESEARCH MACHINES

LOGO

For Stand-Alone Disc Systems

REFERENCE MANUAL

PN 10945

Research Machines LOGO Reference Manual

PN 10945

Release 1, March 1983

Copyright (c) 1983 by Research Machines Limited

All rights reserved. Copies of this publication may be made by customers exclusively for their own use, but otherwise no part of it may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language without the prior written permission of Research Machines Ltd., Post Office Box 75, Mill Street, Oxford OX2 0BW, England. Telephone Oxford (0865) 249866.

The policy of Research Machines Limited is one of continuous development and improvement of its products and services, and the right is therefore reserved to revise this document or to make changes in the computer software it describes without notice. Research Machines make every endeavour to ensure the accuracy of the contents of this document but do not accept liability for any error or omission.

The original labelled distribution disc is regarded as the only proof of purchase and must be produced in order to qualify for an update at a reduced rate. Keep it safe and always work from copies.

Reference is occasionally made to other Research Machines publications. Such publications are included with each computer system but in case of difficulty copies may be ordered from Research Machines at the address above.

Additional copies of this publication may also be ordered from Research Machines. Please ask for "Research Machines LOGO Reference Manual for Stand-Alone Disc Systems".

RESEARCH MACHINES LOGO

FOR STAND-ALONE DISC SYSTEMS

Reference Manual

Research Machines Ltd.
P O Box 75, Oxford.

Copyright 1983 Research Machines Ltd

CONTENTS

Section

1. Introduction
   1.1 Terminology

2. The LOGO system
   2.1 Organisation of the system
   2.2 How to start the system
   2.3 Finishing a session

3. Using Research Machines LOGO
   3.1 User input
   3.2 Commands
   3.3 Data types: numbers, words and lists
   3.4 Errors while using LOGO

4. User defined procedures
   4.1 The editor
   4.2 Using the editor
   4.3 Some points about procedures
   4.4 Understanding the flow of control
       in user-defined procedures

5. LOGO PRIMITIVES
   5.1 TURTLE GRAPHICS
   5.2 MATHEMATICAL
   5.3 CONDITIONAL
   5.4 BOOLEAN
   5.5 LIST HANDLING
   5.6 GENERAL
   5.7 CONTROL
   5.8 EDITING
   5.9 FILING
   5.10 DEBUGGING

6. The Filing System
   6.1 Repairing unintentionally changed files
   6.2 LOGO command files
   6.3 The LOGO configuration file

7. Some examples of LOGO

Appendix A. Comparison of Research Machines
   and Terrapin LOGO
Appendix B. About the printers

Index

Copyright 1983 Research Machines Ltd

INTRODUCTION

This manual describes the Research Machines LOGO system and language. It is aimed at users with at least a little computing experience, though perhaps none of LOGO.

LOGO is an interactive, procedure-based list processing and graphics handling language, designed for educational use. It was invented at Bolt, Beranek and Newman Inc., in Boston, in the late 1960's, and it has since been promoted widely by them and by LOGO groups at the Massachusetts Institute of Technology, the University of Edinburgh and elsewhere.

The most distinctive feature of LOGO is the 'Turtle', a notional animal that the user can direct around the screen using body-centred commands such as FORWARD and LEFT, and which can draw a line behind it as it moves. The ease with which the Turtle can be controlled to produce elaborate and pleasing drawings, either directly or through user-defined procedures, has made LOGO a powerful tool for teaching spatial concepts and relationships, ideas about object representation, mathematical concepts in geometry and arithmetic, programming concepts and techniques and problem solving skills, particularly the skill of 'top-down' analysis of problems.

Those interested in the educational aspects of LOGO should consult:

> MINDSTORMS: CHILDREN, COMPUTERS AND POWERFUL IDEAS. By Seymour Papert. Brighton, Sussex: Harvester Press, 1980.
>
> TURTLE GEOMETRY. By Hal Abelson and Andrea DiSessa. Cambridge, Massachusetts: M. I. T. Press, 1981.
>
> BYTE, THE SMALL SYSTEMS JOURNAL: LOGO ISSUE. Byte 7, 8, August 1972. Published by BYTE Publications Inc., 70 Main Street Peterborough NH 03458, United States.

Research Machines LOGO is based on Edinburgh LOGO, a dialect of the language which has been used during the last ten years to support research into the role of LOGO in school mathematics teaching. Reflecting as it does the Edinburgh LOGO group's concern that a programming language should be simple to use, reliable and friendly, Edinburgh LOGO emerged from an intensive program of experimentation with the language, carried out over three years. It involved forty 12-15 year old children of average ability who spent the order of 2,500 terminal hours working with the language which was altered in various ways to make it easier to learn. Edinburgh LOGO differs from other LOGO systems in a number of respects e.g. choice of names for primitives, wording of error messages, omission of infix operators. Appendix A contains a brief comparison of Research

Machines LOGO with Terrapin LOGO, the latter being based on the dialect of the language in use at Massachusetts Institute of Technology.

Sections 2 to 5 of this manual describe how to set up the system, and describe the fundamentals of using LOGO.

There is another document, "The Hitch Hiker's Guide to LOGO", which gives a tutorial introduction to the use of the language.

1.1 Terminology

The Research Machines LOGO system, as seen by the user, consists of over a hundred "primitives", i.e. system procedures, that the user can employ for such tasks as controlling the Turtle, multiplying numbers and printing characters on the display screen, also known as the "screen".

It allows the user to build his own "procedures", indistinguishable from primitives when in use. LOGO is designed to manipulate three types of data: "words", which consist of character strings, "numbers" and "lists", (explained in detail in section 3.3) which are ordered sets of numbers, words and other lists. Both procedures and primitives require zero or more "inputs", each a number, word or list, and may return a "result", again a number, word or list, for use as an input to other procedures or primitives.

The user talks to the system by typing "commands" which the system obeys as far as it can, reporting the error if it cannot fully obey for any reason.

Copyright 1983

THE LOGO SYSTEM

This section of the Manual describes the organisation of the system, and how to start and finish a session.

2.1 Organisation of the system

The LOGO system consists of a single eight-inch or five-and-a-quarter-inch floppy disc, which contains the following:

1) The LOGO system. This is held in the file LOGO.COM.

2) An error messages file, ERRMSGS0.LGO, This means that the disc must be in the drive if the user wants the full error messages, though there is a set of brief messages which LOGO will use if it cannot find the file.

3) LOGO procedure files, in which the user has stored LOGO procedures. When used for the first time, there will be no procedure files on the disc. Once the user has used the KEEP command, there will be at least one file whose name ends in ".LGP". This is a "procedure file". It may be on the LOGO system disc, or on any other disc. The filing system will also create files whose names end in ".BAK" and, exceptionally, in ".$$$". Files whose name ends in ".$$$" can be deleted. This is explained further in the notes on the Filing System (see section 6).

4) A news file, NEWS.LGO. The contents of this file will be printed out every time LOGO is started. It is a good way to send a message to several users.

5) Possibly a configuration file, CONFIG.LGO. The contents of this file will be read by LOGO to give it information about the type of printer in use, and other information normally communicated by "flags" (See section 2.2).

2.2 How to start the system

Switch on the computer or press the RESET button. Insert a CP/M System disk in drive A and the LOGO disk in drive B. Type

B

to boot the system. The computer will print

```
RESEARCH MACHINES
56K CP/M VERSION 1.4
```

or something very similar. Log in to drive B by typing

A>B:

followed by the RETURN key. You should only type the underlined characters; the computer will print the rest. The computer responds by printing

B>

Then start LOGO by typing its name, again followed by RETURN:

B>LOGO

You can, of course, put CP/M onto your LOGO disc and use only drive A. Refer to your CP/M Users Guide for details.

The following flags may be used. (A flag is typed after the command LOGO and a space):

-M This forces LOGO to ignore the error messages file, and use its internal messages. This flag is mainly intended for maintenance. The error messages file is used by default.

-N This suppresses the printing of the "news" at the start of the session. The news is printed by default.

-PA The -PA flag tells LOGO that the printer is an Anadex 9500 or 9501. You must use this flag, if this is the case.

-PE The -PE flag tells LOGO that the printer is an Epson. This is the default printer type. If you intend to use a printer it must be initialized in the normal way before starting LOGO.

In addition the user can specify a file name from which input will be taken instead of from the keyboard. The default file type is ".LGC" (for LOGO Command). Only one file name can be specified.

If LOGO is reading input from a file, pressing ESC on the keyboard will stop LOGO reading the file, and return it to its normal state.

Examples:

B>LOGO -PA

This example will run LOGO using the Anadex printer for graphics.

B>LOGO -N DEMO

```
B>LOGO B:DEMO -N
```

Both these examples will start LOGO, suppressing the news, and taking input from the file DEMO.LGC instead of from the keyboard. In the case of the second example, the file DEMO.LGC is specified to be on disc B of the computer.

In situations where the use of flags might be confusing, it is possible to build up a disc which will always start LOGO with a particular set of arguments. Details are in Section 6.

When LOGO is fully started and ready to accept a command, it displays the "Waiting" prompt

```
W:
```

a cursor and a flashing Turtle.

2.3 Finishing a session

After saving any wanted procedure in a file (see section 6), the command

```
GOODBYE
```

finishes the session, and returns the user to CP/M.

There is no harm in switching off without saying GOODBYE providing you have saved any procedures you want to use again, but it is advisable to remove the discs before doing so.

USING RESEARCH MACHINES LOGO

LOGO commands are used to control the turtle and to manipulate LOGO data items, namely numbers, words and lists. Commands, made from combinations of about a hundred primitives described in section 5, are typed on a keyboard. The user is able to extend indefinitely the set of commands that the system will accept.

The primitives BUILD, CHANGE and INSPECT each make use of an editor to allow the user to define, modify or merely examine his own procedures. This is dealt with in section 4. This section is devoted to the rules of user input, command construction and a description of how the system reacts to errors.

3.1 User Input

The computer can be in one of three states : Waiting, Defining or Executing. The letter W: at the left hand side of the screen indicates that the system is waiting for the user to type in a command. Normally, what the user types is echoed immediately on the screen at the position of the "cursor" (the small rectangle). Characters typed when the system is not expecting them are lost. Any lower case alphabetic characters are converted to upper case, so it is easy to distinguish LOGO's error messages which are largely in lower case. Normally spaces are used as separators, but commas may also be used in the same way. For example, the list

```
[160 120]
```

might be typed as

```
   [160,120]
or [160, 120]
or [   160   120   ]
```

If the user types a command, the system will try to execute it as soon as the Return key is pressed. A command that does not fit into a single line can be extended over additional lines, since it is not an error to type across the right hand edge of the screen. Printing resumes on the next line and the command will execute normally. The general syntax of a command is described in section 3.2 below.

In the Waiting state, some keys have special effects. They are listed below. In the list, the symbol "CTRL/" means "hold down the control key" while pressing the key with which the symbol is paired.

ESCAPE Always stops everything and returns the system to a state of waiting for the user to type a new command. On a few machines this key may be marked "ALT. MODE".

DEL Deletes the last character typed. This may be labelled "RUB OUT" on some keyboards.

RETURN Tells LOGO to execute the command which has just been typed.

CTRL/A (Autopage) The "autopage toggle" is a standard feature. If a program produces more than a screenful of output, the cursor stops at the bottom of the screen, and the program will not resume until the user presses a key. Typing CTRL/A at any time will suspend this feature. Typing CTRL/A again will reinstate it.

CTRL/K (Kill) Deletes the whole of the command typed so far, and moves the cursor back to the prompt.

CTRL/Z This has the same effect as ESCAPE (see above). In addition it gives a "symbolic dump" showing everything that was happening when the CTRL/Z was processed.

The names of procedures currently executing are printed, together with the values of their inputs and the values of any variables created by the "NEW" command. The procedure most recently started is dumped first. Variables are dumped in order of their creation. Uninitialised variables dump as "?".

After the display of procedure information, global variables are dumped.

The dump can be interrupted by pressing ESCAPE or CTRL/Z.

The primitive BUILD is used to build a new procedure. BUILD is followed by a word, which must begin with a letter - the example 2A is not legal. When the RETURN key is pressed LOGO changes from the Waiting state to the Defining state. A "box" appears on the screen, indicating that the editor is ready for use. The name of the procedure is written in the box's upper left hand corner, with the cursor located at its right hand end. When the editor is in use, various keys have special effects. Some move the cursor around the screen; others insert or delete lines and others move the procedure text up or down. They are listed and explained in section 4.1. The ESCAPE key is pressed to return the user to the Waiting state. The user may also change the text of a previously constructed procedure. In the Waiting state, the user types the commmand CHANGE followed by a word (the name of the procedure changed). When the RETURN key is pressed LOGO changes from the Waiting state to the Defining state. The procedure's text is displayed inside the editing box. The special keys are used to make the necessary changes. As

before, pressing ESCAPE returns the user to the Waiting state.

3.2 Commands

A command typed in response to the W: prompt consists of the name (it is not prefixed by any special character) of any procedure or primitive that does not return a result, followed by any necessary inputs. LOGO uses Polish notation for its command syntax, meaning that any "operator" (a primitive or procedure) is followed by its "operands" (its inputs). The LOGO system assumes that any name not prefixed by a special character (such as a colon or quote mark), and not enclosed in square brackets (see section 3.3), is the name of a procedure or primitive to be executed in the course of the command. The following examples illustrate this idea:

```
W: FORWARD 50
```

This is a complete command.

```
W: PRINT 'HELLO
```

This is also a complete command. LOGO treats PRINT as the name of a primitive or procedure. It finds that PRINT needs one input. The quote mark before HELLO indicates that what follows should be treated simply as a word rather than as the name of something to be called, and so LOGO will just print the word HELLO.

```
W: PRINT ADD 3 4
```

In this command, LOGO calls PRINT, which needs one input. It calls ADD to provide a result to be this input, and ADD takes two inputs in order to construct the result 7. Thus the number 7 is printed.

```
W: PRINT ADD 3 ADD 4 5
```

Here PRINT calls ADD, which needs two inputs. ADD finds one input, 3, and then finds the second ADD command. In turn, this needs two inputs, 4 and 5. The result, 9, provides the second input for the first ADD command, making in effect ADD 3 9. Hence the number 12 is printed.

The full syntax of a command is:

command:- name of procedure or primitive that does not return a result, followed by such inputs as necessary. A complete command may be enclosed in round brackets, for legibility or to distinguish commands when using AND,

WHILE, REPEAT or IF.

input: number, word, or list, or name of a procedure or primitive that does return a result. (section 3.3 describes the construction of numbers, words and lists.) An input can be enclosed in round brackets for the sake of legibility.

3.3 Data types: numbers, words and lists

The basic types of data that LOGO uses are numbers, words and lists. Each is represented on the terminal in a distinctive way:

Number: A number consists of one or more digits, optionally including a decimal point. It may be prefixed by a minus sign "-". There must be no separators anywhere within it. The following are valid numbers:

```
347
-347.
347.098765
```

The Research Machines LOGO system can only cope with seven significant figures. Numbers with more than this may not be held with complete accuracy. The absolute value of a number may not be greater than about 10↑38 or less than about 10↑-20 (excepting zero).

Word: A word is denoted by a quote mark at the start, followed immediately by a string made up of zero or more characters chosen from the following:

```
A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z 0 1 2 3 4 5 6 7 8 9
! " $ & % * + - . / ; = ? @ ↑ _ ` { | }
```

In addition any string starting with 0-9, - or . and which cannot be converted to a number is a word.

These are valid words:

```
'ATILLA
'23
'5FINGERS
'-=AND*SO_ON
'
2A    (cannot be converted to a number)
```

List: A list is either dynamic or static. A static list, which is normal, is an ordered series of numbers, words and lists enclosed in square brackets. (On some machines square brackets are displayed as arrowheads.)

In a static list, a word is not begun with a quote mark, the square brackets ensuring that it is not treated as a procedure to be run. These are examples of static lists:

```
[ABC D 12]
[A [[B C] [D]] E F]
[12.5 -8.9 HELLO]
[ ]
```

A dynamic list is an ordered sequence of numbers, words with quote mark, lists, and names of primitives or procedures returning a result, followed by their inputs. These are enclosed in angle brackets. When the command containing the dynamic list is obeyed, the procedures are called to provide the appropriate elements. These are examples of dynamic lists:

```
< 'A 'B 'C >
< ADD 3 ADD 4 5 >
< SQT 49 >
< >
```

The first example is equivalent to [A B C]. The second example is equivalent to [12]. The third is equivalent to [7]. The fourth is equivalent to [].

Any list, or any expression whose value is a list, may be followed by the list pointer # and an integer. The value of

```
list # integer
```

is the "integer"-th element of "list". For example,

```
MAKE 'C [THIS IS A STATIC LIST]
PRINT :C # 4
```

will print the word STATIC.

Note that you can construct expressions that seem ambiguous. For example, REST is a primitive which, when given a list input returns the list formed by omitting the first element. The expression

```
PRINT REST [[A B] [C D]] # 1
```

will print [B] rather than [C D], because the # operation is always done first.

The list pointer may also be used to change the value of one element of a list stored in a variable. (See the "MAKE" command in section 5.6).

Various primitives may be used to assemble and take apart lists. See section 5.5.

3.4 Errors while using LOGO

If any error is detected while the system is executing a command, an error message is printed and the system returns to Waiting for the user to type his next command. The normal error messages are fairly detailed, and should tell the user just what has gone wrong. For example, if the user types the command

```
W: FORWARD 'FRED   (giving a word instead of a number)
```

the system will not move the turtle but will print the error message

```
LOGO can't do "FORWARD" in that command
because input to "FORWARD" must be a number.
```

If the command is part of a procedure definition, e.g. if procedure TRIPE contains

```
.
.
FORWARD 'FRED
.
.
```

then the error message would be

```
LOGO can't do "FORWARD" at level 1 in TRIPE
because input to "FORWARD" must be a number.
```

If the user typed

```
W: PRINT ADD 2 (omitting a second input)
```

then the error message would be

```
LOGO can't do "ADD" in that command
because "ADD" needs two inputs but it only found one.
```

In general, if an error is detected while executing a user-defined procedure, then the error message will name the command in question and print the current "level" (the depth of procedure nesting) as well as explaining what the error actually is.

The error messages exemplified above, being long, can be tiresome to the user with some experience of the system. For this reason there is a primitive TERSE which switches the system over to printing brief error messages instead. For example, the short form of the message above generated by FORWARD 'FRED is

```
LOGO error - Input isn't a number.
"FORWARD FRED"
```

If there is no copy of the error message file on the LOGO system disc, or LOGO was started with the -M flag, then a small set of less specific messages will be used.

If any error occurs while the turtle is moving, the turtle is stopped where it is. The position of the arrowhead accurately represents the turtle's heading and co-ordinates, and these can be determined by using the primitive WHERE (see section 5.1).

Copyright 1983

USER DEFINED PROCEDURES

The most useful feature of LOGO is the ease with which the user can define his own procedures to extend the set of commands that LOGO will accept. Such procedures can be used in exactly the same way as the primitives built into LOGO - they can take any (fixed) number of parameters, and return a result or not, as desired. A procedure consists of two main parts: the title line and the body.

> The title line contains the name of the procedure and a series of words naming the inputs, so that the commands in the body of the procedure can refer to their values.
>
> The body contains a series of commands which will be obeyed when the procedure is run.

For example, the following procedure draws a triangle, of a size specified by the input. The meaning of the colon sign : is explained in section 5.6 under "VALUE".

```
TRIANGLE 'SIZE
FORWARD :SIZE       (forward by the value of the input
RIGHT 120           (then right by 120 degrees
FORWARD :SIZE       (and so on ...
RIGHT 120
FORWARD :SIZE
RIGHT 120
```

The first line (TRIANGLE 'SIZE) is the title line, and the rest is the body.

There are six primitives for dealing with procedures -

i) BUILD — to create a procedure.

ii) CHANGE — to amend an existing procedure.

iii) SCRAP — to delete one or more from memory.

iv) RESTORE — to undo all that the most recent CHANGE did.

v) INSPECT — to examine a procedure without changing it.

vi) PROCEDURES — to display the names of all existing procedures.

BUILD, CHANGE and INSPECT each take a word as input (see section 5.8). They call the editor to allow the user to provide, change or examine the text making up the whole definition of the procedure in question.

4.1 The editor

The editor cleans the text display, suppresses the graphics and constructs a "box" in the top two thirds of the screen. Commands from the keyboard control the position of the cursor within the box, until the user finishes editing by pressing ESCAPE or CTRL/Z.

Rather than think of the box as a border round a sheet of paper, it should be thought of as a window onto a roll of paper (one per procedure) which stretches vertically above and below the window. The roll can be moved up or down to provide more space when the user is defining a long procedure. Alternatively, it can be moved up or down, to expose previously written text which is hidden from view below or above the window.

4.2 Using the editor

When the editor is invoked by BUILD, only the name of the procedure appears in the top left hand corner of the window, with the cursor positioned at the right hand end of the name. During typing, if the cursor reaches the right hand border of the window, it moves to the next line (taking any non-complete word with it). The * characters making up the border at the end of the line are replaced by angle bracket characters, to remind the user that they are not separate lines. If the editor is invoked by CHANGE, the window is filled with the text of the procedure and the cursor is positioned at the beginning of the line following the last line of the procedure's text. If there are more lines of text than will fit in the window, there will be lines still to come into the bottom of the window. Under these circumstances, the cursor is positioned at the right hand end of the bottom line of text. As example, the command

```
CHANGE 'TRIANGLE
```

will lead to a display like this:

```
****************************************
* TRIANGLE 'SIZE                       *
* FORWARD :SIZE                        *
* RIGHT 120                            *
* FORWARD :SIZE                        *
* RIGHT 120                            *
* FORWARD :SIZE                        *
* RIGHT 120                            *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
****************************************
```

During editing, when words are inserted into existing text, words to the right of the insertion are moved to the right automatically. Similarly, when words are deleted from the text, words to the right of the deletion are moved to the left automatically.

When the editor is invoked by INSPECT, all the keys on the keyboard are ignored except for those which move the cursor.

The special keys available for use during BUILD and CHANGE are listed below. Those which perform special functions in the Waiting state perform similar functions while editing.

DEL — Deletes the character to the left of the cursor and moves the cursor to the left one space unless it is already at the left hand border of the window (when nothing happens).

ESCAPE — Causes the editor to store the contents of the window as part (or all) of the procedure definition, and then finish. It wakes up the graphics and returns the system to the Waiting state, providing that there is enough space in the memory to store the complete procedure definition.

RETURN — Moves cursor to start of next line. If the cursor is on the bottom line of the window, the text is scrolled up one line.

CTRL/A — (Autopage) This key is ignored during editing.

CTRL/B — (Begin) Moves the cursor to the beginning of the text. If the beginning of the text is above the window, the text is scrolled down.

CTRL/C — Moves the cursor to the top left corner of the window.

CTRL/D — (Down) Moves the cursor down one line

CTRL/E — Moves the cursor to the end of the text. If the end of the text is below the window, the text is scrolled up.

CTRL/G — Moves the cursor and text up eight lines. If the text goes outside the window, the cursor is positioned on the top line.

Copyright 1983

CTRL/H (Help) Describes the available control characters: each time CTRL/H is pressed, a subset of the total is displayed above the window.

CTRL/K (Kill) Erases the line of text on which the cursor is standing: more text may appear in the bottom of the window.

CTRL/L (Left) Moves the cursor left one position.

CTRL/R (Right) Moves the cursor right one position.

CTRL/S (Split) Splits the line on which the cursor is standing into two lines at the cursor position.

CTRL/T Moves the cursor and text down eight lines. If the text goes outside the window, the cursor is positioned on the bottom line.

CTRL/X (eXcise) Deletes the character on which the cursor is standing. Characters on the cursor's right move left one place to fill the gap.

CTRL/U (Up) Moves the cursor up one line.

CTRL/Z Cancels editing session. and returns the user to the Waiting state. The BUILD or CHANGE command has no effect. The word "Cancelled" appears below the window.

CTRL/Z will close the window, but all work will be lost. This is the only option available if LOGO runs out of space during editing.

The title of the procedure may be changed during BUILD or CHANGE. During BUILD, this means that the title originally specified as input to BUILD will be ignored, and the new one substituted. During CHANGE, it means that the procedure whose title was originally specified will be unchanged by the editing, and a new procedure with the new name will be created. It is illegal to change the title to a new title if the new title is in use already.

After CHANGE (but not BUILD) a "back up" copy of the procedure is created. This means that the previous version of the procedure can be recovered.

Copyright 1983

The procedure RESTORE will delete the new copy of the procedure most recently CHANGEd, and replace it with the back up copy, thereby cancelling the effect of the most recent CHANGE command.

The back up copy is destroyed by the next BUILD or CHANGE. If a CHANGE session is cancelled with CTRL/Z, no back up is created. If the PROCEDURES command is run when a back up copy of a procedure is available, the name appears on the screen twice, marked on the second occurrence by "(old)".

4.3 Some points about procedures

The only things that will cause an error are:

i) missing or unintelligible title line
ii) running out of space

and these are detected only when "ESC" is pressed. If there is an error, the error message appears just below the window and the editor is re-entered. If there is no error, there is a short pause while the procedure is written into memory, and then the window is closed.

Procedures can call each other, and can also call themselves, although there is a limit to how deeply nested procedure calls can be. The names of the inputs can be the same as those of other variables, and those variables will be invisible while the procedure is actually being executed. When a procedure is being executed, the system tries to locate variables as required in the following order:

1) as a local variable in the procedure, created by NEW (see section 5.6)
2) as an input to the procedure
3) as a local variable of the calling procedure
4) as an input to the calling procedure

.
.
and so on, back through all the calling procedures
.
.

n) lastly, as a global variable

4.4 Understanding the flow of control in user-defined procedures

If errors happen when a user-defined procedure is being run, the following explanation may help in the process of debugging:

Think of the system as having an unlimited number of copies of every procedure and a vast number of "index fingers" for keeping track of where it is. When a procedure is called:

i) the system gets a copy of its text
ii) it makes new variables for the inputs, naming them as in the title line and giving them values as specified by the

call

iii) it begins to move a finger from left to right along each line, from one line to the next, executing primitives as it encounters them.

If it encounters a call to another procedure,

i) it moves its finger along to gather up all the arguments
ii) it keeps that finger there to mark the place to return to after executing the new procedure
iii) it gets a copy of the new procedure's text

N.B. this could be the same as the calling procedure - a situation known as a recursive call

iv) it begins to move a new finger from left to right along each line from one line to the next, executing primitives until it reaches the end, i.e., the end of the last line, or STOP or ESCAPE or RESULT.
v) it returns to the place marked by the previous procedure (ii above)

If a procedure is marked for WALKING (see section 5.10) then each line of text is printed before it is executed.

PRIMITIVES AVAILABLE

In this list the meanings of the words in lower case in the "syntax" sections are:

command..... Any LOGO command

condition..... an expression whose value is TRUE or FALSE

drivename..... a quoted one-letter word, between 'A and 'M inclusive

expression..... any LOGO expression

filename..... a 1- to 8-letter word made up of 0-9 and A-Z

integer..... a number whose value is an integer

list..... an expression whose value is a list

number..... an expression whose value is a number

procedurelist..... a list of procedure names

procedurename..... a word which is the name of a user-defined procedure

word..... an expression whose value is a word

An "American title" is an alternative name for a primitive which LOGO recognises.

Copyright 1983

5.1 Turtle Graphics

NAME..... ARCL

SYNTAX..... ARCL number number

EXAMPLE(S)..... ARCL 30 360

Moves the Turtle through an arc to the left. First input specifies radius, and second the angle of the arc.

NAME..... ARCR

SYNTAX..... ARCR number number

EXAMPLE(S)..... ARCR 30 360

Moves the Turtle through an arc to the right. Like ARCL.

NAME..... BACKWARD

SYNTAX..... BACKWARD number

EXAMPLE(S)..... BACKWARD 50

ABBREVIATION..... BD

Moves the Turtle backwards in a straight line.

NAME..... CENTRE

AMERICAN TITLE.... CENTER

SYNTAX..... CENTRE

ABBREVIATION..... CT

Returns the Turtle to the centre of the screen. The centre is at (160, 95).

NAME..... CLEAN

SYNTAX..... CLEAN

ABBREVIATION..... CL

Cleans the screen.

NAME..... DRAWING

SYNTAX..... DRAWING

Sets 4-line scroller, so removing text from the drawing area.

NAME..... EDGES

SYNTAX..... EDGES

When EDGES is in force the Turtle may be moved only within the bounds 0 <= x <= 319 and 0 <= y <= 191. (i.e. within the screen area) See NOEDGES.

NAME..... FAST

SYNTAX..... FAST

Sets Turtle speed to the higher of its two values. See SLOW.

NAME..... FORWARD

SYNTAX..... FORWARD number

EXAMPLE(S)..... FORWARD 50

ABBREVIATION..... FD

Moves the Turtle forwards in a straight line.

NAME..... HIDE

SYNTAX..... HIDE

Hides the Turtle's position.. Restored by REVEAL.

NAME..... LABEL

SYNTAX..... LABEL expression

EXAMPLE(S)..... LABEL 'P

Writes characters in graphics mode. The value of "expression" appears on the screen with the leftmost character near the Turtle. The colour of the writing is the Turtle pencil colour. The Turtle is not moved.

NAME..... LEFT

SYNTAX..... LEFT number

EXAMPLE(S)..... LEFT 90

ABBREVIATION..... LT

Rotates the Turtle leftwards (i.e. anticlockwise) the required number of degrees.

NAME..... LIFT

SYNTAX..... LIFT

"Lifts the Turtle's pencil off the paper" i.e., it ceases to draw on the screen as it moves. The opposite is PENCIL, below.

NAME..... MIX

SYNTAX..... MIX

Text and drawing share the whole screen. (Default.)

NAME..... NOEDGES

SYNTAX..... NOEDGES

The Turtle can be moved anywhere within the region bounded by -32757 <= x <= 32757 and -32757 <= y <= 32757. (i.e., outside the screen area) See EDGES.

NAME..... PENCIL

SYNTAX..... PENCIL integer

EXAMPLE(S)..... PENCIL 1

Makes the turtle draw. Pencil numbers are:
0 - none,
1 - white,
2 - red,
3 - green.
The default is 1. These will appear as shades of grey on monochrome monitors.

NAME..... PICTURE

SYNTAX..... PICTURE

Copies the picture currently being displayed to the printer. Normally LOGO sends a sequence which makes sense only to the Epson MX-80. The "Anadex" Model 9500 or 9501 printer can be used instead, provided LOGO is started with the -PA flag. Printers other than the Epson and the Anadex will print garbage. (See Appendix A.)

PICTURE is not interruptible with ESC or CTRL/Z since interrupting it would leave the printer in graphics mode.

NAME..... REVEAL

SYNTAX..... REVEAL

Marks the position of the Turtle by a flashing arrowhead (default.)

NAME..... RIGHT

SYNTAX..... RIGHT number

EXAMPLE(S)..... RIGHT 180

ABBREVIATION..... RT

Rotates the Turtle rightwards (i.e. clockwise) the required number of degrees.

NAME..... RUBBER

AMERICAN TITLE.... ERASER

SYNTAX..... RUBBER

Erases lines in a drawing. Synonymous with PENCIL 0.

NAME..... SETH

SYNTAX..... SETH number

EXAMPLE(S)..... SETH 0

Changes Turtle heading to "number" degrees. At 0 degrees, the Turtle is facing right; heading increases anti-clockwise.

NAME..... SETX

SYNTAX..... SETX number

EXAMPLE(S)..... SETX 20

Changes the Turtle's X co-ordinate to "number", giving horizontal teleportation without drawing.

NAME..... SETY

SYNTAX..... SETY number

EXAMPLE(S)..... SETY 100

Changes Turtle's Y co-ordinate to "number" giving vertical teleportation without

drawing.

NAME..... SLOW

SYNTAX..... SLOW

Sets Turtle speed to the slower of its two values. (Default.) See FAST.

NAME..... TEXT

SYNTAX..... TEXT

Switches off drawing leaving text.

NAME..... WHERE

SYNTAX..... WHERE

EXAMPLE(S)..... PRINT WHERE

Returns a list of 3 numbers whose significance is as follows:
WHERE # 1 is the Turtle's current X co-ordinate.
WHERE # 2 is the Turtle's current Y co-ordinate.
WHERE # 3 is the Turtle's current heading.

NAME..... WIPE

SYNTAX..... WIPE

Cleans text off the screen, leaving just a W: prompt in the bottom left hand corner.

5.2 Mathematical

NAME..... ADD

SYNTAX..... ADD number number

EXAMPLE(S)..... PRINT ADD 2 3

NAME..... COS

SYNTAX..... COS number

EXAMPLE(S)..... PRINT COS 45

Cosine of number of degrees. The above example will not be correct unless decimal places are set first. See DECIMALS below.

NAME..... DECIMALS

SYNTAX..... DECIMALS integer

EXAMPLE(S)..... DECIMALS 3

Specifies the number of digits which will be printed after the decimal point when a LOGO number is printed. Arguments in the range 0-6 are legal.

NAME..... DIVIDE

SYNTAX..... DIVIDE number number

EXAMPLE(S)..... PRINT DIVIDE 21 7

ABBREVIATION..... DIV

Performs division: in this case, 21 is divided by 7.

NAME..... MULTIPLY

SYNTAX..... MULTIPLY number number

EXAMPLE(S)..... PRINT MULTIPLY 5 10

ABBREVIATION..... MUL

NAME..... PICK

SYNTAX..... PICK integer

EXAMPLE(S)..... PRINT PICK 25

Prints a random integer between 1 and "integer".

NAME..... RANDOM

SYNTAX..... RANDOM

Returns a random number between 0 and 0.999999

NAME..... REMAINDER

SYNTAX..... REMAINDER number number

EXAMPLE(S)..... PRINT REMAINDER 25 6

ABBREVIATION..... REM

Remainder when first number is divided by second.

NAME..... SHARE

SYNTAX..... SHARE number number

EXAMPLE(S)..... PRINT SHARE 25 6

Integer quotient when first number is divided by second.

NAME..... SIN

SYNTAX..... SIN number

EXAMPLE(S)..... PRINT SIN 45

Sine of number of degrees. Like COS, see above.

NAME..... SQT

SYNTAX..... SQT number

EXAMPLE(S)..... PRINT SQT 16

Square root of number.

NAME..... SUBTRACT

SYNTAX..... SUBTRACT number number

EXAMPLE(S)..... PRINT SUBTRACT 72 60

ABBREVIATION..... SUB

Performs subtraction: in this case, 60 is subtracted from 72.

NAME..... TAN

SYNTAX..... TAN number

EXAMPLE(S)..... PRINT TAN 45

Tangent of number of degrees.

5.3 Conditional

NAME..... EMPTYQ

SYNTAX..... EMPTYQ expression

EXAMPLE(S)..... IF EMPTYQ :L THEN...

ABBREVIATION..... EMQ

TRUE if its input is
[] (the empty list) or
FALSE otherwise.

NAME..... EQUALQ

SYNTAX..... EQUALQ expression expression

EXAMPLE(S)..... IF EQUALQ :I 6 THEN...

ABBREVIATION..... EQQ

TRUE if its two inputs are equal, FALSE otherwise. Numbers are equal if they would be printed as equal strings.

NAME..... GREATEREQUALQ

SYNTAX..... GREATEREQUALQ expression expression

EXAMPLE(S)..... IF GREATEREQUALQ :X 10 THEN...

ABBREVIATION..... GEQ

TRUE if its first input is greater than or equal to its second, FALSE otherwise. GREATEREQUALQ cannot compare lists. If its first input is a number, the second must be a number; if its first input is a word, the second must be a word.

NAME..... GREATERQ

SYNTAX..... GREATERQ expression expression

EXAMPLE(S)..... IF GREATERQ :X 10 THEN...

Copyright 1983

ABBREVIATION..... GRQ

TRUE if its first input is greater than its second, FALSE otherwise. The restriction on comparing lists, and on comparing words with numbers, applies to GREATERQ. (See GREATEREQUALQ).

NAME..... KEYQ

SYNTAX..... KEYQ

TRUE if a key has been pressed, FALSE otherwise. Once a key has been pressed, KEYQ is true until the key is read with ASK or KEY. If LOGO is reading from a command file KEYQ is always true.

NAME..... LESSEQUALQ

SYNTAX..... LESSEQUALQ expression expression

EXAMPLE(S)..... IF LESSEQUALQ :X 10 THEN...

ABBREVIATION..... LEQ

TRUE if its first input is less than or equal to its second, FALSE otherwise. Restricted in the same way as GREATEREQUALQ.

NAME..... LESSQ

SYNTAX..... LESSQ expression expression

EXAMPLE(S)..... IF LESSQ :X 10 THEN...

ABBREVIATION..... LSQ

TRUE if its first input is less than its second, FALSE otherwise. Restricted in the same way as GREATEREQUALQ.

NAME..... LISTQ

SYNTAX..... LISTQ expression

EXAMPLE(S)..... IF LISTQ :R THEN...

ABBREVIATION..... LQ

TRUE if its input is a list, FALSE otherwise.

NAME..... NUMBERQ

SYNTAX..... NUMBERQ expression

EXAMPLE(S)..... IF NUMBERQ :I THEN...

ABBREVIATION..... NQ

TRUE if its input is a number, FALSE otherwise.

NAME..... WORDQ

SYNTAX..... WORDQ expression

EXAMPLE(S)..... IF WORDQ :X THEN...

ABBREVIATION..... WQ

TRUE if its input is a word, FALSE otherwise.

NAME..... ZEROQ

SYNTAX..... ZEROQ expression

EXAMPLE(S)..... IF ZEROQ :I THEN...

ABBREVIATION..... ZQ

TRUE if its input is zero, FALSE otherwise. It is not an error for "expression" to be a word or a list, but in these cases the result will always be FALSE.

5.4 Boolean

NAME..... BOTH

SYNTAX..... BOTH condition condition

EXAMPLE(S)..... IF BOTH EMPTYQ :L1 EMPTYQ :L2 THEN...

TRUE if both its input conditions are true. Both conditions are always evaluated. There is no word "AND" between the two inputs.

NAME..... EITHER

SYNTAX..... EITHER condition condition

EXAMPLE(S)..... IF EITHER EMPTYQ :L1 EMPTYQ :L2 THEN...

TRUE if either, or both, of its input conditions are true. Both conditions are always evaluated.

NAME..... NOT

SYNTAX..... NOT condition

EXAMPLE(S)..... IF NOT ZEROQ :X THEN...

TRUE if its input condition is FALSE; FALSE if its input condition is TRUE.

NAME..... XOR

SYNTAX..... XOR condition condition

TRUE if one input condition is TRUE and the other FALSE (exclusive OR).

Copyright 1983

5.5 List handling

NAME..... FIRST

SYNTAX..... FIRST word
FIRST list

EXAMPLE(S)..... PRINT FIRST 'HUMPTY
PRINT FIRST [HUMPTY DUMPTY SAT ON A WALL]

Returns the first letter of a word or the first element of a list. FIRST cannot be used with the empty word or list.

NAME..... JOIN

SYNTAX..... JOIN word word
JOIN list list

EXAMPLE(S)..... PRINT JOIN 'GORGON 'ZOLA
PRINT JOIN [THERE WAS AN OLD WOMAN] [WHO LIVED IN A SHOE]

Joins two words or two lists into one.

NAME..... PUTFIRST

SYNTAX..... PUTFIRST expression list

EXAMPLE(S)..... PRINT PUTFIRST 1 [2 3 4]

ABBREVIATION..... PF

Returns a list whose first element is "expression" and the rest of whose elements are those of "list".

NAME..... PUTLAST

SYNTAX..... PUTLAST list expression

EXAMPLE(S)..... PRINT PUTLAST [1 2 3] 4

ABBREVIATION..... PL

Returns a list whose last element is "expression".

Copyright 1983

NAME..... REST

SYNTAX..... REST word
REST list

EXAMPLE(S)..... PRINT REST [WHAT IS YOUR DOG DOING]
PRINT REST 'BEETLE

Returns a word whose letters are all but the first of "word", or a list whose elements are all but the first of "list".

REST cannot be used with an empty word or empty list.

5.6 General

NAME..... ASK

SYNTAX..... ASK expression

EXAMPLE(S)..... MAKE 'NAME ASK [WHAT IS YOUR NAME]

The value of the expression is printed, followed by a question mark. If the value is a list, it is printed without its outer square brackets. The system pauses to allow the user to type a reply. The reply may not contain the characters < > ( ) ' : # or unmatched square brackets. When the user presses RETURN, the reply is returned as a list.

While typing a reply to a question asked by ASK,

ESC interrupts the program,

DEL deletes the last character typed,

CTRL/K deletes everything typed so far, back to the '?',

CTRL/Z interrupts the program, giving a symbolic dump.

NAME..... COPY

SYNTAX..... COPY procedurename
COPY procedurelist

EXAMPLE(S)..... COPY 'MYPROGRAM
COPY [MYPROGRAM YOURPROGRAM HISPROGRAM]

Sends a copy of named program(s) to the printer. Procedures are printed separated by two blank lines. The COPY is completed with a form feed. COPY [ ] sends a form feed only to the printer.

NAME..... COUNT

SYNTAX..... COUNT expression

EXAMPLE(S)..... PRINT COUNT [WHAT HO JEEVES]

Returns a number as its result:

(a) if "expression" is a number, it returns the number of digits to the left of the decimal point (minimum 1, as in 0.6);

(b) if "expression" is a word, it returns the number of its characters;

(c) if "expression" is a list, it returns the number of its elements.

NAME..... GOODBYE

SYNTAX..... GOODBYE

Stops LOGO. Return the user to CP/M.

NAME..... KEY

SYNTAX..... KEY

Returns the value of the next key pressed on the keyboard. KEY waits for a key, if necessary. If LOGO is reading from a command file, then KEY returns the next character from the file. Non-printing characters are returned as '?'.

NAME..... MAKE

SYNTAX..... MAKE word expression
MAKE word # integer expression

EXAMPLE(S)..... MAKE 'X 50
MAKE 'X # 4 100

Assigns a new value to the variable called "word". If "word" does not exist, a global variable with the name "word" is created.

The alternative syntax "word # integer expression" may be used when the variable "word" exists and its value is a list comprising at least "integer" elements. Its effect is to change the value of the "integer"-th element to "expression". The example **MAKE 'X # 4 100** will set the fourth element of X to 100.

NAME..... NEW

SYNTAX..... NEW word
NEW list

EXAMPLE(S)..... NEW 'Y
NEW [COUNT TEMP LINE_SIZE]

Creates a local variable with the name "word". The variable ceases to exist once the procedure containing the "NEW" instruction stops. If the input is a list, up to 10 variables are created, their names being the elements of the list. Variables created by "NEW" are initialised to the value "?".

NAME..... PAUSE

SYNTAX..... PAUSE number

EXAMPLE(S)..... PAUSE 10

The system pauses for approximately "number" seconds.

NAME..... PRINT

SYNTAX..... PRINT expression

EXAMPLE(S)..... PRINT :X

NAME..... RESULT

SYNTAX..... RESULT expression

EXAMPLE(S)..... RESULT :X

Specifies the result of a procedure.

NAME..... RUN

SYNTAX..... RUN word [expression...]

EXAMPLE(S)..... RUN 'WHERE
RUN 'LEFT 90

Runs the program whose name is "word". If the program requires inputs, these follow "word" (example 1). If the program returns a result, its result is returned as the result of RUN (example 2).

NAME..... SAY

SYNTAX..... SAY expression

EXAMPLE(S)..... SAY [HELLO WORLD]

The value of the expression is printed on the screen. If the value is a list, it is printed without its outermost square brackets.

NAME..... STOP

SYNTAX..... STOP

Stops the procedure in which the STOP occurs, handing back control to the procedure which invoked it.

NAME..... VALUE

SYNTAX..... VALUE word

EXAMPLE(S)..... PRINT VALUE 'X
PRINT :X

ABBREVIATION..... :

Returns the value of the variable whose name is "word". The colon sign must

immediately prefix "word". Note that VALUE VALUE 'X is legal but ::X is not. Thus the colon is not a true abbreviation for VALUE.

5.7 Control

NAME..... AND

SYNTAX..... command AND command

EXAMPLE(S)..... FORWARD 50 AND LEFT 90

Links two commands together.

NAME..... DO

SYNTAX..... DO command UNTIL condition

EXAMPLE(S)..... DO MYPROGRAM UNTIL GREATERQ :X 10

Always paired with "UNTIL". The command is executed at least once, and repeated until the condition is true.

NAME..... ESCAPE

SYNTAX..... ESCAPE

Stops the program in which the ESCAPE occurs and stops any program which invoked it.

NAME..... IF

SYNTAX..... IF condition THEN command
IF condition THEN command ELSE command
(IF condition THEN expression ELSE expression)

EXAMPLE(S)..... IF LESSQ :X 50 THEN FORWARD :X
IF LESSQ :X 50 THEN FORWARD :X ELSE FORWARD 50
FORWARD (IF LESSQ :X 50 THEN :X ELSE 50)

Enclosing the whole IF expression in parentheses will save trouble, especially when you have nested IF statements. When IF is used to return a result, an ELSE clause is obligatory, though a missing ELSE clause could only be caught as an error if it were needed

when being run.

NAME..... REPEAT

SYNTAX..... REPEAT integer command

EXAMPLE(S)..... REPEAT 5 FORWARD 60 AND LEFT 144

The argument "integer" is evaluated only once. REPEAT 0 is legal.

NAME..... WHILE

SYNTAX..... WHILE condition command

EXAMPLE(S)..... WHILE LESSQ :X 10 MYPROG
WHILE 'TRUE MYPROG

Command is executed zero or more times. The second example shows how to repeat a command indefinitely.

5.8 Editing

NAME..... BUILD

SYNTAX..... BUILD procedurename

EXAMPLE(S)..... BUILD 'MYFUNCTION

Switches the system into edit state.

NAME..... CHANGE

SYNTAX..... CHANGE procedurename

EXAMPLE(S)..... CHANGE 'MYFUNCTION

"procedure" must exist already. Switches the system into edit state.

NAME..... INSPECT

SYNTAX..... INSPECT procedurename

EXAMPLE(S)..... INSPECT 'MYFUNCTION

"procedurename" must exist already. Switches the system into edit state, with the keyboard locked except for ESCAPE CTRL/G CTRL/T CTRL/B CTRL/E and CTRL/Z function keys.

NAME..... PROCEDURES

SYNTAX..... PROCEDURES

Lists the procedures currently defined. If there is a back-up copy present, it is marked "old".

NAME..... RESTORE

SYNTAX..... RESTORE

Cancels last editing session, if back-up still exists.

NAME..... SCRAP

SYNTAX..... SCRAP procedurename
SCRAP procedurelist

EXAMPLE(S)..... SCRAP 'MYFUNCTION
SCRAP [SQUARE TRIANGLE CIRCLE]

Deletes the named procedure(s) from memory.

5.9 Filing

NAME..... DISC

AMERICAN TITLE.... DISK

SYNTAX..... DISC drivename

EXAMPLE(S)..... DISC 'C

Filing system uses specified disc instead of CP/M's current disc.

NAME..... FILENAME

SYNTAX..... FILENAME filename

EXAMPLE(S)..... FILENAME 'MYNEWLIB

Changes the active file from DEFAULT.LGP (or its current name). The only secondary name permitted is ".LGP" which LOGO supplies.

NAME..... FILES

SYNTAX..... FILES

Prints the names of all files on the current disc whose names end in ".LGP".

NAME..... GET

SYNTAX..... GET procedurename
GET procedurelist

EXAMPLE(S)..... GET 'SUPERPROGRAM
GET [THISPROG THATPROG]

Fetches program(s) from file into memory.

NAME..... INITIALISE

AMERICAN TITLE.... INITIALIZE

SYNTAX..... INITIALISE

ABBREVIATION..... INIT

Initialises the disc system, then re-selects as current disc the disc that was current when the session was started. Hence it prevents disc errors caused by exchanging the disc on which procedure files are kept.

NAME..... INDEX

SYNTAX..... INDEX

Prints the names of the procedures in the active file.

NAME..... KEEP

SYNTAX..... KEEP procedurename
KEEP procedurelist

EXAMPLE(S)..... KEEP [THISPROG THATPROG]

Keeps named procedure(s) in the active file.

NAME..... LOSE

SYNTAX..... LOSE procedurename
LOSE procedurelist

EXAMPLE(S)..... LOSE 'THISPROG
LOSE [THISPROG THATPROG]

Deletes named procedure(s) from the active file.

5.10 Debugging

The meaning of "bugging", "walking" and "tracing" is as follows:

1) If a variable is marked for "bugging", its value is printed each time it changes.

2) If a procedure is marked for "tracing", its name and the values of its inputs are printed out each time it is called. and the value of any result which it returns is printed when it stops.

3) If a procedure is marked for "walking", before each line is executed it is printed out, and LOGO waits for the user to press a key before continuing. The keys which may be pressed are as follows:

   ESCAPE Stops the program immediately.

   RET Takes one step.

   CTRL/C Walking ceases for this procedure at this level.

   CTRL/F Walking ceases and program finishes normally.

   CTRL/Z Stops the program, giving a symbolic dump.

   Any other key causes a helpful message.

NAME..... BUG

SYNTAX..... BUG word
BUG list

EXAMPLE(S)..... BUG 'X
BUG [X Y Z]

Marks the named variable(s) for bugging.

NAME..... TERSE

SYNTAX..... TERSE

Specifies that the set of short error messages shall be used.

NAME..... TRACE

SYNTAX..... TRACE word
TRACE list

EXAMPLE(S)..... TRACE 'MYPROG
TRACE [PROG_A PROG_B PROG_C]

Marks the named procedure(s) for tracing.

NAME..... UNBUG

SYNTAX..... UNBUG word
UNBUG list

EXAMPLE(S)..... UNBUG 'X
UNBUG [X Y Z]

Cancels the effect of BUG

NAME..... UNTRACE

SYNTAX..... UNTRACE word
UNTRACE list

EXAMPLE(S)..... UNTRACE 'MYPROG
UNTRACE [PROG_A PROG_B PROG_C]

Cancels the effect of TRACE

NAME..... UNWALK

SYNTAX..... UNWALK word
UNWALK list

EXAMPLE(S)..... UNWALK 'MYPROG
UNWALK [PROG_A PROG_B PROG_C]

Cancels the effect of WALK

NAME..... VERBOSE

SYNTAX..... VERBOSE

Specifies that the set of long error messages shall be used. (Default.)

Copyright 1983

NAME..... WALK

SYNTAX..... WALK word
WALK list

EXAMPLE(S)..... WALK 'MYPROG
WALK [PROG_A PROG_B PROG_C]

Marks the named procedure(s) for walking.

LOGO FILING SYSTEM

The LOGO filing system enables a user to store procedures he has written in a disc file, or to retrieve procedures from a disc file.

Procedures are stored in, or read from, the current "active file". When LOGO is first started, the active file is the file DEFAULT.LGP. This means that a user can use KEEP and GET without understanding the technicalities of the system. To specify a different active file the command FILENAME is given, followed by a quoted name of 1-8 characters. Only the characters A-Z and 0-9 are valid in a filename. The secondary file name .LGP will be supplied by LOGO automatically.

The file specified need not exist at the time the FILENAME command is given. It will be created, if necessary, by the first KEEP command given.

The command DISC enables the user to specify that the active file shall be on a disc other than the currently logged disc. Discs must be specified as a quoted one-letter word A-D. For example, to specify that the active file is on Disc B

```
DISC 'B
```

The command FILES will produce a list of all the files on the current disc whose secondary filename is .LGP, and which are presumed to be LOGO program files. The current disc is either the currently logged disc, or the disc most recently specified by a DISC command.

6.1 Repairing unintentionally changed files

During the first phase of the command, <file>.LGP is read in and <file>.$$$ is written out. Once this is successfully completed, <file>.LGP is renamed to <file>.BAK, and <file>.$$$ is renamed to <file>.LGP. Hence, during the first phase of the command, hitting ESC will leave <file>.LGP unchanged. If too late...

This is the way to recover the file as it was before KEEP or LOSE was executed.

1) Say GOODBYE to LOGO, without issuing any further KEEP or LOSE commands

2) Issue the command

```
ERA DEFAULT.LGP
```

to CP/M. (If a FILENAME command was given, the file will have some other primary name, but the secondary name will still be .LGP.)

3) Issue the command

```
REN DEFAULT.LGP=DEFAULT.BAK
```

(or whatever the file is called) to CP/M. Now the file is restored.

6.2 LOGO command files

A LOGO command file is a file of LOGO commands which LOGO will read as its input, instead of reading from the keyboard, until ESC or CTRL/Z is pressed or end of file or a GOODBYE command is reached.

A command file can be prepared using a text file editor, such as "TXED". The secondary file name should normally be ".LGC".

Control characters may be included in the command file by representing them as a 2-character sequence. The first character of the sequence is "↑" (Ascii 5E hex.), and the second is the appropriate capital letter. For example, the control character ↑R (cursor right) will appear in a command file as the two character sequence "↑R". The two character sequence "↑↑" will be read as "↑".

Note that ESCAPE is represented by ↑[.

Inclusion of BUILD and CHANGE commands in a command file is possible but inadvisable. File the procedure(s) in a file and use FILENAME and GET instead. If it is necessary to include BUILD or CHANGE in a command file, remember the position of the cursor when entering the editor. The following sequence is correct:

```
BUILD 'TRIANGLE
'LENGTH
REPEAT 3 FORWARD :LENGTH AND LEFT 120
↑[TRIANGLE 70
```

To run the command file, state its name when starting LOGO (see section 2.2).

6.3 The LOGO configuration file

The LOGO configuration file, if it exists, holds any number of valid LOGO flags separated by carriage returns or by spaces, and possibly the name of one LOGO command file. Text must be in capital letters. The name of the configuration file must be CONFIG.LGO. The configuration file will be read before the command line arguments. It is, therefore, possible to over-ride it.

For example, suppose the file CONFIG.LGO contains the text

```
-PA
-N
```

then starting LOGO by just typing LOGO will put the -PA and -N options into effect.

Starting LOGO by typing

```
LOGO -PE +N
```

cancels the -N option and specifies the Epson printer. Note the use of '+' to cancel the corresponding '-' option. Valid cancellations are:

+M use the error message file

+N display the news

A file name specified on the command line replaces any specified in the configuration file.

SOME EXAMPLES OF LOGO

Some simple LOGO procedures and commands are given below, with suitable comments, in order to give the reader some flavour of using the system.

```
W: FORWARD 75
W: RIGHT 90
W: FORWARD 75 AND RIGHT 90 AND FORWARD 75 AND RIGHT 90
W: FORWARD 75 AND RIGHT 90
```

These four commands will make the turtle draw a square of side 75.

```
W: REPEAT 4 FD 75 AND RT 90
```

This is a shorter way to draw a square.

```
W: BUILD 'SPIRAL

SPIRAL 'SIDE 'ANGLE
IF GREATERQ :SIDE 166 THEN ESCAPE
FORWARD :SIDE AND RIGHT :ANGLE
SPIRAL ( ADD :SIDE 5 ) :ANGLE
```

This is a recursive procedure for drawing patterns. It needs two inputs: the length of the first side, and an angle to turn. ESCAPE stops the procedure (and the recursion) if the side is bigger than 166. Otherwise it makes the turtle move forward and turn right by the the amounts specified, a final line SPIRAL (ADD :SIDE 5) :ANGLE making it all happen again but with the side length increased by 5 units.

```
W: SPIRAL 5 122
```

This draws a 'rotating triangle' pattern, using the above procedure.

Copyright 1983

```
W: BUILD 'HELLO

HELLO
NEW 'REPLY
PRINT 'HELLO
MAKE 'REPLY ASK [WANT SOME CHOCOLATE]
IF EQUALQ FIRST :REPLY 'YES THEN SAY [SO WOULD I] AND STOP
IF EQUALQ FIRST :REPLY 'NO THEN SAY [WHAT A PITY]
PAUSE 5 AND SAY [IT WAS DELICIOUS]
```

This procedure is a whimsical example of the use of ASK to get an answer from the user. It prints WANT SOME CHOCOLATE? and stores the reply as a list in the variable named REPLY. If the reply began with YES then SO WOULD I is printed and the procedure ends, without returning a result. If the reply began with NO, then the procedure prints WHAT A PITY, waits five seconds, prints IT WAS DELICIOUS, and then ends. Bugs: the program will crash if the reply contains no words. Replies beginning other than with YES or NO are not handled properly.

```
W: BUILD 'ARCSIN

ARCSIN 'LOWBOUND 'HIGHBOUND 'X
NEW 'Z AND MAKE 'Z DIV ADD :LOWBOUND :HIGHBOUND 2
IF EQQ :X ( SIN :Z ) THEN RESULT :Z
RESULT IF GRQ :X (SIN :Z) THEN ARCSIN :Z :HIGHBOUND :X ELSE \
ARCSIN :LOWBOUND :Z :X
```

```
W: BUILD 'ARCSINE

ARCSINE 'Q
RESULT ARCSIN -90 90 :Q
```

These two procedures can be used to find the arcsine of a number, by an interval halving method. The ARCSINE procedure exists only to call the ARCSIN procedure with certain initial values of the inputs. ARCSIN calls itself recursively taking better and better guesses at the answer. Eventually it returns a result to ARCSINE, which returns a result

(the same one in effect) to the procedure that called it. It would be better if ARCSINE did two further things:

a) checked first that the input lay between -1 and 1
b) set the number of decimals output to 6 (by DECIMALS 6)

before calling ARCSIN, and reset it appropriately afterwards. This would ensure that the most accurate answer was obtained as otherwise the test in line 20 of ARCSIN could succeed when the guess was still fairly far wrong. For example, if the number of decimals output is 0, then EQQ 0.4 SIN 0 returns 'TRUE, so that these two procedures would return the angle 0 as the arcsine of 0.4!

```
W: BUILD 'APPLY

APPLY 'PROCNAME 'L
IF EMQ :L THEN STOP
RUN :PROCNAME FIRST :L
APPLY :PROCNAME REST :L
```

This procedure calls the procedure named by the first input once for each element of the list given as the second input, using each element in turn as the input to it. For example,

```
APPLY 'SCRAP [TEST1 TEST2 TEST3]
```

will cause the three procedures named TEST1, TEST2 and TEST3 to be scrapped, and the command

```
APPLY 'TRACE [PROC1 PROC2 PROC3]
```

will mark those three procedures for tracing.

APPENDIX A
A COMPARISON OF RESEARCH MACHINES LOGO AND TERRAPIN LOGO

The comparison between dialects is based on nearest equivalents. To make an exact comparison, look at detailed descriptions given in the relevant manuals.

DRAWING COMMANDS

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
ARCL	n n			
ARCR	n n			
BACKWARD (BD)	n	BACK (BK)	n	
CENTRE (CT)		HOME		
CLEAN (CL)		CLEARSCREEN (CS)		
DRAWING		FULLSCREEN		
EDGES		NOWRAP		
FAST				
FORWARD (FD)	n	FORWARD (FD)	n	
HIDE		HIDETURTLE (HT)		
LABEL	nwl			
LEFT (LT)	n	LEFT (LT)	n	
LIFT		PENUP (PU)		
MIX		SPLITSCREEN		
NOEDGES				
PENCIL	n	PENDOWN (PD)		
PICTURE				
REVEAL		SHOWTURTLE (ST)		
RIGHT (RT)	n	RIGHT (RT)	n	
RUBBER				
SETH	n	SETH	n	
SETX	n	SETX	n	Not teleportation
SETY	n	SETY	n	Not teleportation
		SETXY	n n	
SLOW				
TEXT		NODRAW (ND)		
WHERE	=> l	XCOR	n	
		YCOR	n	
		HEADING	n	
WIPE		CLEARTEXT		
		BACKGROUND (BG)	n	Colour 0-6
		CURSOR		Positions "input cursor"

DRAWING COMMANDS (continued)

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
		TOWARDS	n n	Gives heading Returns list
		TURTLESTATE (TS)		
		WRAP		

ARITHMETIC

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
ADD	n n => n	+		Infix
COS	n => n	COS	n => n	Degrees
DECIMALS	n (0-6)			
DIVIDE (DIV)	n n => n	/		Infix
MULTIPLY(MUL)	n n => n	*		Infix
PICK	n => n	RANDOM	n => n	Integer 0 to n-1
RANDOM				
REMAINDER	n n => n	REMAINDER	n n => n	
SHARE	n n => n	QUOTIENT	n n => n	Integer division
SIN	n => n	SIN	n => n	degrees
SQT	n => n	SQRT	n => n	
SUBTRACT(SUB)	n n => n	-		Infix
TAN	n => n			
		ATAN	n n => n	Nearly the same as TOWARDS
		INTEGER	n => n	
		RANDOMISE		
		ROUND	n => n	

WORDS and LISTS

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
FIRST	nwl => nwl	FIRST	l => l	
JOIN	wl wl => wl	SENTENCE (SE)	wl wl..	
				=> Concatenates
		WORD	w w.. =>	
				w Concatenates
PUTFIRST (PF)	nwl l => l	FPUT	nwl l => l	
PUTLAST (PL)	l nwl => l	LPUT	nwl l => l	Order of input changed
REST	nwl => nwl	BUTFIRST (BF)	l => l	
		BUTLAST (BL)	l => l	
		LAST	l => l	
		LIST	l l.. => l	Similar to dynamic lists

PROCEDURES

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
BUILD	w	TO	w	No quote
CHANGE	p	EDIT (ED)	w	Note no quote
INSPECT	p	TEXT	w => l	Goes with DEFINE
PROCEDURES		PRINTOUT (PO)	w	No quote, qualifiers OK
RESTORE				
SCRAP	pl	ERASE (ER)	w	No quote, qualifiers OK
		DEFINE	w l	Non-interactive

PROCEDURES (continued)

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
		END		Separator in editor

BOOLEAN

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
BOTH	t t => t	ALLOF	t t.. => t	
EITHER	t t => t	ANYOF	t t.. => t	
NOT	t => t	NOT	t => t	
XOR	t t => t			

CONDITIONALS

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
EMPTYQ (EMQ)	l => t			
EQUALQ (EQQ)	nwl nwl => t	=	nwl nwl => t	
GREATEREQUALQ (GEQ)	nw nw => t			
GREATERQ (GRQ)	nw nw => t	>	n n => t	
KEYQ	=> t	RC?	=> t	
LESSEQUALQ (LEQ)	nw nw => t			
LESSQ (LQ)	nw nw => t	<	n n => t	
LISTQ (LQ)	nwl => t	LIST?	nwl => t	
NUMBERQ (NQ)	nwl => t	NUMBER?	nwl => t	
WORDQ (WQ)	nwl => t	WORD?	nwl => t	

CONDITIONALS (continued)

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
ZEROQ (ZQ)	nwl => t	THING?	w => t	Tests for instantiation

CONTROL

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
DO cmd UNTIL t				
ESCAPE		TOPLEVEL		
IF t THEN cmd		IF t THEN action		
IF t THEN cmd ELSE cmd		IF t THEN action ELSE action		
IF t THEN nwl ELSE nwl => nwl		IFTRUE action (IFT)		
		IFFALSE action (IFF)		
		TEST	t	Used with IFTRUE
REPEAT	n cmd	REPEAT	n l	
RESULT	nwl	OUTPUT (OP)	nwl	
STOP stops procedure		STOP		
WHILE	t cmd			
		GO	w	Labels at start of line

GENERAL

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
AND	cmd			
ASK	l => l	REQUEST	=> l	
COPY	pl			
COUNT	nwl => n			
GOODBYE		GOODBYE		
KEY	=> nw	READCHARACTER (RC)		=> nw
MAKE	w nwl	MAKE	w nwl	
NEW	wl			
PAUSE	n			
PRINT	nwl	PRINT (PR)	nwl..	
		PRINT1	nwl..	No final ‹return›
RUN	p			
SAY	l			
VALUE (:)	w => nwl	THING (:)	w => nwl	
		ASCII	w => n	Input must be a character
		CHAR	n => w	Returns a character
		CLEARINPUT		Flushes input buffer
		OUTDEV	n	Select Apple card
		PADDLE	n => n	Input 0-3, output 0-255
		PADDLEBUTTON	n => t	

FILING

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
FILENAME	w			
FILES		CATALOG		
GET	pl	READ	w	Reads whole file
INDEX				
INITIALISE				
KEEP	pl	SAVE	w	Saves the lot in a file
LOSE	pl	ERASEFILE	w	
		READPICT	w	
		SAVEPICT	w	

DEBUGGING

Primitive	Inputs and Result	Primitive	Inputs and Result	Comments
BUG	wl			
TERSE				short error messages
TRACE	pl			
UNBUG	wl			
UNTRACE	pl			
UNWALK	pl	NOTRACE		
VERBOSE				(default)
WALK	pl	TRACE		
		CONTINUE (CO)		..after PAUSE
		PAUSE		Invokes interpreter

APPENDIX B

ABOUT THE PRINTER

LOGO can send copies of the picture currently being displayed to either of two printer types: the Epson MX series and the Anadex DP-9500 or DP-9501.

This Appendix describes the exact sequence of characters into which the picture is translated. The notation

<x>

has been adopted to represent the character whose hexadecimal code is x; common names for Ascii codes are given in parentheses.

The "aspect ratio" of a printer is the ratio of the lengths of two lines, one horizontal and one vertical (in that order), produced by identical Turtle movements in horizontal and vertical directions.

The Epson MX series

This type of printer is selected by default, or by starting LOGO with the '-PE' flag.

The Epson has an "aspect ratio" of about 1.00 : 0.83. Pictures printed on it will appear too wide.

To send a picture to the Epson, LOGO first sends the sequence

<1B> <33> <15> (ESC '3' Nak)

This sets the line-feed to about 0.97 inches (7-dot vertical spacing).

The picture is transmitted as a set of lines, each 7 dots high. A line consists of the sequence

<1B> <4B> <40> <01> (ESC 'K' '@' Soh)

which sets single density bit image mode for the next 320 bytes, followed by 320 "graphics bytes", followed by

<0D> <0A> (CR LF)

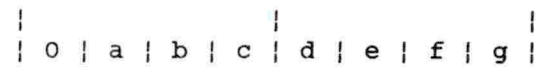
The last graphics line is followed by the sequence

<0C> <1B> <40> (FF ESC '@')

which causes a form feed and then causes the printer to re-initialise itself from the current switch settings.

Copyright 1983

Each "graphics byte" is made up of 8 bits, the leftmost always zero. (Although the Epson senses the leftmost bit, it cannot be used because of the limitations of the parallel port.) For example the first byte transmitted is that representing the points in the top left hand corner of the picture, whose x co-ordinates are all 0 and whose y co-ordinates range from 191 to 185 inclusive in the format



The bit marked '0' is always clear. Bit "a" is 1 if the point (0, 191) is painted and 0 otherwise. Bit "b" represents the point (0, 190) and so on.

One final point to note is that some versions of the Epson printer print the hash sign '#' (Ascii 23 hex.) as a pound sign. To change this, ensure that the internal DIP switch pins 1-7, 2-1 and 2-2 are set to:

1-7	ON
2-1	ON
2-2	ON

This selects the American character set. (The Epson operation manual explains how to do this.) There are no side effects.

The Anadex printer

Specifying -PA as a "flag" when starting LOGO selects this printer type.

The Anadex has an "aspect ratio" of about 1.00 : 1.02. This is so close to unity that no distortion is apparent in the picture.

At the start of a picture, the sequence

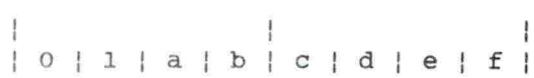
<0D> <0A> <1C> (CR LF FS)

is sent. This puts the Anadex printer into its "graphics mode".

The picture is transmitted as a set of lines each 6 dots high. Each line consists of 320 "graphics bytes", followed by

<36> ('6').

Each "graphics byte" is a byte in the range <40> to <7F>. The rightmost six bits only are significant. As an example, the very first byte carries information about the six points in the top left hand corner of the picture, whose x co-ordinates are all 0 and whose y co-ordinates range from 191 to 186 in this format:



The bit marked 0 is always clear; that marked 1 always set. "a" is 1 if the point (0, 191) is painted and 0 otherwise; "b" represents the point (0,190) and so on.

The picture is followed by

<36> <1D> ('6' GS).

Then, in order to re-position the printer at the next perforation in the paper, the character

<0A> (LF)

is sent 48 times.

The Anadex loses its information about form length, top-of-form and skip-over-perforation when it is put into graphics mode. To re-establish these, the sequence

<1B> <48> <1B> <25> <30> <36> <1B> <34> <30> <36> <36>
(ESC 'H' ESC '%' '0' '6' ESC '4' '0' '6' '6')

is sent. This specifies that the printer adopt

a vertical line spacing of 6 lines/inch,
a skip-on-perforation of 6 lines,
a form length of 11 inches.

It is unfortunately impossible to send the Anadex any sequence which will cause the switch settings to be re-read. What is sent is a common set of defaults, but if a different length of paper is used or if the 8 lines/inch vertical spacing is preferred, the printer must be turned off and then on again before proceeding.

Copyright 1983

..... 3.5
: 3.3, 4.1, 5.20, 5.28

active file 5.27, 6.1
ADD 5.8
AND 5.22
ARCL 5.2
ARCR 5.2
ASK 5.17

BACKWARD 5.2
BOTH 5.14
BUG 5.28
BUILD 3.1, 3.2, 4.1, 4.2, 4.4, 5.24

CENTRE 5.2
CHANGE 3.1, 3.2, 4.1, 4.2, 4.4, 4.5, 5.24
CLEAN 5.3
command 3.3
command file 6.2
configuration file 6.2
COPY 5.17
COS 5.8
COUNT 5.18
CTRL/A 3.2, 4.3
CTRL/B 4.3
CTRL/B 5.24, 4.3
CTRL/C 5.28
CTRL/D 4.3
CTRL/E 4.3, 5.24
CTRL/F 5.28
CTRL/G 4.3, 5.24
CTRL/H 4.4
CTRL/K 3.2, 4.4, 5.17
CTRL/L 4.4
CTRL/R 4.4
CTRL/S 4.4
CTRL/T 4.4, 5.24
CTRL/U 4.4
CTRL/X 4.4
CTRL/Z 3.2, 4.2, 4.4, 4.5, 5.17, 5.24, 5.28, 5.5, 6.2

debugging 4.5
DECIMALS 5.8
DEL 3.2, 4.3, 5.17
DISC 5.26, 6.1
DIVIDE 5.8
DO 5.22
DRAWING 5.3
dynamic list 3.5

EDGES 5.3
EITHER 5.14
EMPTYQ 5.11
EQUALQ 5.11

errors 3.6
ESC 2.2, 4.3, 4.5, 5.17, 5.5, 6.2
ESCAPE 3.1, 3.2, 3.3, 4.2, 4.3, 4.6, 5.22, 5.24, 5.28, 6.2

FAST 5.3
FILENAME 5.26
FILES 5.26
filing system 2.1, 6.1
FIRST 5.15
flags 2.1, 2.2, 2.3, 6.2
FORWARD 5.3

GET 5.26
GOODBYE 2.3, 5.18, 6.2
GREATEREQUALQ 5.11
GREATERQ 5.11

HIDE 5.4

IF 5.22
INDEX 5.27
INITIALISE 5.27
INSPECT 3.1, 4.1, 5.24

JOIN 5.15

KEEP 2.1, 5.27
KEY 5.18
KEYQ 5.12

LABEL 5.4
LEFT 5.4
LESSEQUALQ 5.12
LESSQ 5.12
LIFT 5.4
list 1.2, 3.4
LISTQ 5.12
LOSE 5.27

MAKE 5.18
MIX 5.4
MULTIPLY 5.8

NEW 3.2, 5.19
news 2.1, 2.2, 2.3, 6.3
NOEDGES 5.5
NOT 5.14
number 1.2, 3.4
NUMBERQ 5.13

PAUSE 5.19
PENCIL 5.5
PICK 5.9
PICTURE 5.5
PRINT 5.19

printer 2.1, 2.2, 5.17, 5.5, 6.3
procedures 1.2, 2.1, 2.3, 4.1, 6.1
PROCEDURES 4.1, 4.5, 5.24
PUTFIRST 5.15
PUTLAST 5.15

RANDOM 5.9
REMAINDER 5.9
REPEAT 5.23
REST 5.16
RESTORE 4.1, 4.5, 5.24
RESULT 4.6, 5.19
REVEAL 5.5
RIGHT 5.6
RUBBER 5.6
RUN 5.20

SAY 5.20
SCRAP 4.1, 5.25
SETH 5.6
SETX 5.6
SETY 5.6
SHARE 5.9
SIN 5.9
SLOW 5.7
SQT 5.10
static list 3.5
STOP 4.6, 5.20
SUBTRACT 5.10

TAN 5.10
TERSE 3.6
TEXT 5.7
TRACE 5.29
turtle 3.1

UNBUG 5.29
UNTRACE 5.29
UNWALK 5.29

VALUE 4.1, 5.20
VERBOSE 5.29

W: 2.3, 3.1, 3.3
WALK 4.6, 5.30
WHERE 5.7
WHILE 5.23
WIPE 5.7
word 1.2, 3.4
WORDQ 5.13

XOR 5.14

ZEROQ 5.13

Copyright 1983

RESEARCH MACHINES

MICROCOMPUTER SYSTEMS